

DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG

```
DDDDDDDD  BBBB BBBB  GGGGGGGG  EEEEEEEEE  XX      XX  TTTTTTTTTT
DDDDDDDD  BBBB BBBB  GGGGGGGG  EEEEEEEEE  XX      XX  TTTTTTTTTT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BBBB BBBB  GG      GG  EEEEEEEE  XX      XX  TT
DD      DD  BBBB BBBB  GG      GG  EEEEEEEE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DD      DD  BB      BB  GG      GG  EE      EE  XX      XX  TT
DDDDDDDD  BBBB BBBB  GGGGGG  EEEEEEEEE  XX      XX  TT
DDDDDDDD  BBBB BBBB  GGGGGG  EEEEEEEEE  XX      XX  TT
                                         ....
                                         ....
                                         ....
                                         ....

LL      LL      SSSSSSSS
LL      LL      SSSSSSSS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SSSSSS
LL      LL      SSSSSS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LL      LL      SS
LLLLLLLLLL  IIIIIII  SSSSSSSS
LLLLLLLLLL  IIIIIII  SSSSSSSS
```

F 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 1
(1)

DBGEXT.REQ

Version: 'V04-000'

*
* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
* ALL RIGHTS RESERVED.
*

* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
* TRANSFERRED.
*

* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
* CORPORATION.
*

* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*

WRITTEN BY

Rich Title

October 1983

MODIFIED BY

Robert Conti

November 2, 1983

Edward Freedman

December 12, 1983

MODULE FUNCTION

This module contains the definitions for the control blocks
that are used in communications between DEBUG and the
ADA multi-tasking run-time system. These same definitions will be
extended for use in communication with the PPA multi-tasking
system and other run-time systems, at a future time.

EXTERNAL CONTROL BLOCK

An "External Control Block" is a data structure that can be used when DEBUG needs to call a routine that is not linked in as part of the DEBUG image.

For example, DEBUG will have commands to support ADA multi-tasking. However, DEBUG has no knowledge of the workings of the ADA multi-tasking system and the data structures that describe tasks. Instead, DEBUG will call a routine in the ADA multitasking system in the course of processing SHOW TASK, SET TASK, or any other command that requires knowledge about tasks.

There will be a single entry point, ADASDBGEXT, in the ADA multitasking system which is called by DEBUG. The External Control Block is the only parameter. Similarly, other multitasking run-time systems will have a single entry point, of the form <facility>SDBGEXT, with the entry point taking an External Control Block as its single parameter. In general, the External Control Block can be used as a means of communication with run-time systems that are not part of DEBUG. For example, in debugging the language SCAN we may want to allow the user to set breakpoints on events such as a SCAN pattern-match. The External Control Block will be the data structure that we use to communicate with the SCAN run-time system.

The DBGEXTSV_FACILITY_ID field identifies which run-time system is being called. The VAX/VMS Facility code is used. Thus, it is assumed that there will be at most one DBGEXT entry point in the run-time code of any facility. Currently, legal values are the facility codes for ADA, PPA, and SCAN. This field may not actually be looked at (if desired, the run-time system may do a sanity check for the right value).

Since there are several functions we want each run-time system to perform for us, there is a DBGEXTSW_FUNCTION_CODE field which specifies which function is to be performed.

All functions return a status code in the DBGEXTSL_STATUS field. For all functions, there is a DBGEXTSL_FLAGS field which can be used as a bitvector of flags. The exact use of these flags depends on the function.

The use of the remaining fields of the data structure depends upon the "FACILITY_ID" field and upon the "FUNCTION_CODE" field.

NOTE: DEBUG makes these calls with ASTs disabled. It is required that the run-time code not reenale ASTs during its execution.

H 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1 Page 3
(3)

0092 0
0093 0
0094 0
0095 0
0096 0
0097 0
0098 0
0099 0
0100 0
0101 0
0102 0
0103 0
0104 0
0105 0
0106 0
0107 0
0108 0
0109 0

The following illustrates the header of an External Control Block.
The fields of an External Control Block are then illustrated
for the case where the "FACILITY_ID" is "ADA".

The following header is common to all External Control Blocks:

0	!unused ! V_FACILITY_ID ! DBGEXT\$W_FUNCTION_CODE !
1	! DBGEXT\$L_STATUS !
2	! (some flags unused) DBGEXT\$L_FLAGS !
3	! reserved for future use !

1 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 4
(4)

0110 0
0111 0
0112 0
0113 0
0114 0
0115 0
0116 0
0117 0
0118 0
0119 0
0120 0
0121 0
0122 0
0123 0
0124 0
0125 0
0126 0
0127 0
0128 0
0129 0
0130 0
0131 0
0132 0
0133 0
0134 0
0135 0
0136 0

The following illustrates the control block when the FACILITY_ID field is "ADA". This control block is used for most functions (some functions, e.g. GET_REGISTERS and SET_REGISTERS use a longer control block, displayed later).

0	unused : V_FACILITY_ID : DBGEXT\$W_FUNCTION_CODE
1	DBGEXT\$L_STATUS
2	(some flags unused) DBGEXT\$L_FLAGS
3	reserved for future use
4	DBGEXT\$L_TASK_VALUE
5	DBGEXT\$L_TASK_NUMBER
6	unused : V_HOLD : V_STATE : DBGEXT\$W_SPECIFIED_FLAGS
7	DBGEXT\$V_PRIORITY
8	DBGEXT\$L_PRINT_ROUTINE
9	DBGEXT\$L_EVENT_ID

J 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 5
(5)

The following fields are present when the "FACILITY_ID" field is
"ADA" and the function code is

DBGEXT\$K_GET_REGISTERS,
DBGEXT\$K_SET_REGISTERS,
DBGEXT\$K_SET_ACTIVE.

For all other functions, the smaller block (without the register fields)
is passed in.

0	unused ! V_FACILITY_ID ! DBGEXT\$W_FUNCTION_CODE
1	DBGEXT\$L_STATUS
2	(some flags unused) DBGEXT\$L_FLAGS
3	reserved for future use
4	DBGEXT\$L_TASK_VALUE
5	DBGEXT\$L_TASK_NUMBER
6	unused ! V_HOLD ! V_STATE ! DBGEXT\$W_SPECIFIED_FLAGS !
7	DBGEXT\$V_PRIORITY
8	DBGEXT\$L_PRINT_ROUTINE
9	DBGEXT\$L_EVENT_ID
10	DBGEXT\$L_R0
11	DBGEXT\$L_R1
12	DBGEXT\$L_R2
13	DBGEXT\$L_R3
14	DBGEXT\$L_R4
15	DBGEXT\$L_R5
16	DBGEXT\$L_R6
17	DBGEXT\$L_R7
18	DBGEXT\$L_R8
19	DBGEXT\$L_R9
20	DBGEXT\$L_R10
21	DBGEXT\$L_R11
22	DBGEXT\$L_AP
23	DBGEXT\$L_FP

K 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 BLISS-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1 Page 6
(5)

0194	0		
0195	00	24	DBGEXT\$\$_SP
0196	00		
0197	00	25	DBGEXT\$\$_PC
0198	00		
0199	00	26	DBGEXT\$\$_PSL
0200	00		
0201	00		
0202	0		

L 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 7
(6)

CONTROL BLOCK FIELDS

FIELD DBGEXT\$HEADER_FIELDS =

```
SET
DBGEXT$W_FUNCTION_CODE      = [ 0, 0, 16, 0],
DBGEXT$V_FACILITY_ID        = [ 0, 16, 12, 0],
! reserved
DBGEXT$L_STATUS              = [ 1, 0, 32, 0],

DBGEXT$L_FLAGS               = [ 2, 0, 32, 0],
DBGEXT$V_ALL                 = [ 2, 0, 1, 0],%((WHAT WILL ALL DO?-tbs))%
DBGEXT$V_FULL                = [ 2, 1, 1, 0],%((explain FULL -tbs))%

DBGEXT$V_PSEUDO_GO           = [ 2, 2, 1, 0],
! Pseudo-go is set by the run-time system on return to DEBUG to
! indicate that DEBUG must do a pseudo-GO to accomplish the function.
! Used only for function SET_ACTIVE (see discussion under SET_ACTIVE).

DBGEXT$V_NO_HEADER           = [ 2, 3, 1, 0]
! Suppresses output of headers on a SHOW_TASK, SHOW_STATISTICS,
! or SHOW_DEADLOCKS.

! reserved                   = [ 0, 4, 28, 0],
! reserved                   = [ 4, 0, 32, 0],
```

TES;

FIELD DBGEXT\$ADA_FIELDS =

```
SET
DBGEXT$L_TASK_VALUE          = [ 4, 0, 32, 0],
DBGEXT$L_TASK_NUMBER         = [ 5, 0, 32, 0],
DBGEXT$W_SPECIFIED_FLAGS     = [ 6, 0, 16, 0],
DBGEXT$V_HOLD_SPECIFIED      = [ 6, 0, 1, 0],
DBGEXT$V_STATE_SPECIFIED     = [ 6, 1, 1, 0],
DBGEXT$V_PRIORITY_SPECIFIED  = [ 6, 2, 1, 0],
! reserved                   = [ 6, 3, 13, 0],
DBGEXT$V_STATE               = [ 6, 16, 4, 0],
DBGEXT$V_STATE_RUNNING       = [ 6, 16, 1, 0],
DBGEXT$V_STATE_READY         = [ 6, 17, 1, 0],
DBGEXT$V_STATE_SUSPENDED     = [ 6, 18, 1, 0],
DBGEXT$V_STATE_TERMINATED    = [ 6, 19, 1, 0],
DBGEXT$V_HOLD                = [ 6, 20, 1, 0],
! reserved                   = [ 6, 21, 11, 0],
DBGEXT$L_PRIORITY            = [ 7, 0, 32, 0],
DBGEXT$V_PRIORITY_00         = [ 7, 0, 1, 0],
DBGEXT$V_PRIORITY_01         = [ 7, 1, 1, 0],
DBGEXT$V_PRIORITY_02         = [ 7, 2, 1, 0],
DBGEXT$V_PRIORITY_03         = [ 7, 3, 1, 0],
DBGEXT$V_PRIORITY_04         = [ 7, 4, 1, 0],
DBGEXT$V_PRIORITY_05         = [ 7, 5, 1, 0],
DBGEXT$V_PRIORITY_06         = [ 7, 6, 1, 0],
DBGEXT$V_PRIORITY_07         = [ 7, 7, 1, 0],
DBGEXT$V_PRIORITY_08         = [ 7, 8, 1, 0],
DBGEXT$V_PRIORITY_09         = [ 7, 9, 1, 0],
DBGEXT$V_PRIORITY_10         = [ 7, 10, 1, 0],
```

M 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 8
(6)

```
0260 0 DBGEXTSV_PRIORITY-11 = [ 7. 11. 1. 0].
0261 0 DBGEXTSV_PRIORITY-12 = [ 7. 12. 1. 0].
0262 0 DBGEXTSV_PRIORITY-13 = [ 7. 13. 1. 0].
0263 0 DBGEXTSV_PRIORITY-14 = [ 7. 14. 1. 0].
0264 0 DBGEXTSV_PRIORITY-15 = [ 7. 15. 1. 0].
0265 0 DBGEXTSV_PRIORITY-16 = [ 7. 16. 1. 0].
0266 0 DBGEXTSV_PRIORITY-17 = [ 7. 17. 1. 0].
0267 0 DBGEXTSV_PRIORITY-18 = [ 7. 18. 1. 0].
0268 0 DBGEXTSV_PRIORITY-19 = [ 7. 19. 1. 0].
0269 0 DBGEXTSV_PRIORITY-20 = [ 7. 20. 1. 0].
0270 0 DBGEXTSV_PRIORITY-21 = [ 7. 21. 1. 0].
0271 0 DBGEXTSV_PRIORITY-22 = [ 7. 22. 1. 0].
0272 0 DBGEXTSV_PRIORITY-23 = [ 7. 23. 1. 0].
0273 0 DBGEXTSV_PRIORITY-24 = [ 7. 24. 1. 0].
0274 0 DBGEXTSV_PRIORITY-25 = [ 7. 25. 1. 0].
0275 0 DBGEXTSV_PRIORITY-26 = [ 7. 26. 1. 0].
0276 0 DBGEXTSV_PRIORITY-27 = [ 7. 27. 1. 0].
0277 0 DBGEXTSV_PRIORITY-28 = [ 7. 28. 1. 0].
0278 0 DBGEXTSV_PRIORITY-29 = [ 7. 29. 1. 0].
0279 0 DBGEXTSV_PRIORITY-30 = [ 7. 30. 1. 0].
0280 0 DBGEXTSV_PRIORITY-31 = [ 7. 31. 1. 0].
0281 0 DBGEXTSL_PRINT_ROUTINE = [ 8. 0. 32. 0].
0282 0 DBGEXTSL_EVENT_ID = [ 9. 0. 32. 0].
0283 0 TES;
0284 0
0285 0 FIELD DBGEXT$REG_FIELDS =
0286 0 SET
0287 0 DBGEXTSL_R0 = [10. 0. 32. 0].
0288 0 DBGEXTSL_R1 = [11. 0. 32. 0].
0289 0 DBGEXTSL_R2 = [12. 0. 32. 0].
0290 0 DBGEXTSL_R3 = [13. 0. 32. 0].
0291 0 DBGEXTSL_R4 = [14. 0. 32. 0].
0292 0 DBGEXTSL_R5 = [15. 0. 32. 0].
0293 0 DBGEXTSL_R6 = [16. 0. 32. 0].
0294 0 DBGEXTSL_R7 = [17. 0. 32. 0].
0295 0 DBGEXTSL_R8 = [18. 0. 32. 0].
0296 0 DBGEXTSL_R9 = [19. 0. 32. 0].
0297 0 DBGEXTSL_R10 = [20. 0. 32. 0].
0298 0 DBGEXTSL_R11 = [21. 0. 32. 0].
0299 0 DBGEXTSL_AP = [22. 0. 32. 0].
0300 0 DBGEXTSL_FP = [23. 0. 32. 0].
0301 0 DBGEXTSL_SP = [24. 0. 32. 0].
0302 0 DBGEXTSL_PC = [25. 0. 32. 0].
0303 0 DBGEXTSL_PSL = [26. 0. 32. 0].
0304 0 TES;
0305 0
0306 0 LITERAL
0307 0 DBGEXT$K_HEADER_SIZE = 4. : Size of header in longwords
0308 0 DBGEXT$K_ADA_SIZE1 = 10. : Size of block for ADA (without regs)
0309 0 DBGEXT$K_ADA_SIZE2 = 27. : Size of block for ADA (with regs)
0310 0 DBGEXT$K_MAX_SIZE = 27. : Max of above sizes
0311 0
0312 0 MACRO
0313 0 DBGEXT$CONTROL_BLOCK = BLOCK [DBGEXT$K_MAX_SIZE]
0314 0 FIELD ( DBGEXT$HEADER_FIELDS,
0315 0 DBGEXT$ADA_FIELDS,
0316 0 DBGEXT$REG_FIELDS ) %;
```

N 15
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 9
(6)

: 0317 0

Generally, multiple priorities and states are valid as input when calling the ADA run time system but are not valid as output values on return from the call. Therefore, the following constants are provided for convenience in setting and testing the contents of the fields DBGEXTSV_STATE and DBGEXTSV_PRIORITY. They define the only possible values of the respective fields when multiple priorities and states are not allowed. Constants for DBGEXTSV_HOLD are provided for completeness.

LITERAL			
DBGEXTSK_MIN_STATE	=	:	%((superfluous? -tbs))%
DBGEXTSK_MAX_STATE	=	:	
DBGEXTSS_STATE	=	4.	! size of DBGEXTSV_STATE
DBGEXTSK_STATE_RUNNING	=	1 ^ 0.	! values for DBGEXTSV_STATE
DBGEXTSK_STATE_READY	=	1 ^ 1.	
DBGEXTSK_STATE_SUSPENDED	=	1 ^ 2.	
DBGEXTSK_STATE_TERMINATED	=	1 ^ 3.	
DBGEXTSS_HOLD	=	1.	! size of DBGEXTSV_HOLD
DBGEXTSK_HOLD	=	1 ^ 0.	! values for DBGEXTSV_HOLD
DBGEXTSS_PRIORITY	=	32.	! size of DBGEXTSV_PRIORITY
DBGEXTSK_PRIORITY_00	=	1 ^ 0.	! values for DBGEXTSV_PRIORITY
DBGEXTSK_PRIORITY_01	=	1 ^ 1.	
DBGEXTSK_PRIORITY_02	=	1 ^ 2.	
DBGEXTSK_PRIORITY_03	=	1 ^ 3.	
DBGEXTSK_PRIORITY_04	=	1 ^ 4.	
DBGEXTSK_PRIORITY_05	=	1 ^ 5.	
DBGEXTSK_PRIORITY_06	=	1 ^ 6.	
DBGEXTSK_PRIORITY_07	=	1 ^ 7.	
DBGEXTSK_PRIORITY_08	=	1 ^ 8.	
DBGEXTSK_PRIORITY_09	=	1 ^ 9.	
DBGEXTSK_PRIORITY_10	=	1 ^ 10.	
DBGEXTSK_PRIORITY_11	=	1 ^ 11.	
DBGEXTSK_PRIORITY_12	=	1 ^ 12.	
DBGEXTSK_PRIORITY_13	=	1 ^ 13.	
DBGEXTSK_PRIORITY_14	=	1 ^ 14.	
DBGEXTSK_PRIORITY_15	=	1 ^ 15.	
DBGEXTSK_PRIORITY_16	=	1 ^ 16.	
DBGEXTSK_PRIORITY_17	=	1 ^ 17.	
DBGEXTSK_PRIORITY_18	=	1 ^ 18.	
DBGEXTSK_PRIORITY_19	=	1 ^ 19.	
DBGEXTSK_PRIORITY_20	=	1 ^ 20.	
DBGEXTSK_PRIORITY_21	=	1 ^ 21.	
DBGEXTSK_PRIORITY_22	=	1 ^ 22.	
DBGEXTSK_PRIORITY_23	=	1 ^ 23.	
DBGEXTSK_PRIORITY_24	=	1 ^ 24.	
DBGEXTSK_PRIORITY_25	=	1 ^ 25.	
DBGEXTSK_PRIORITY_26	=	1 ^ 26.	
DBGEXTSK_PRIORITY_27	=	1 ^ 27.	
DBGEXTSK_PRIORITY_28	=	1 ^ 28.	
DBGEXTSK_PRIORITY_29	=	1 ^ 29.	
DBGEXTSK_PRIORITY_30	=	1 ^ 30.	
DBGEXTSK_PRIORITY_31	=	1 ^ 31.	

C 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1 Page 11
(8)

0375 0
0376 0
0377 0
0378 0
0379 0
0380 0
0381 0
0382 0
0383 0
0384 0
0385 0
0386 0
0387 0
0388 0
0389 0

FACILITY CODES

The following are the possible values of the DBGEXTSV_FACILITY_ID field.
These correspond to the different run-time system we are
communicating with.

ADAS_FACILITY
PPAS_FACILITY
SCNS_FACILITY

QUES Z((-tbs))Z
Do PPA and SCAN have facility mnemonics and codes? Are the
above guesses correct?

FUNCTION CODES

The following are the possible values of the DBGEXTSW FUNCTION_CODE field when the contents of the FACILITY_ID field is ADASFACILITY. These correspond to the functions that the ADA run-time system will be asked to perform.

Summary of the defined function codes

DBGEXTSK_MIN_FUNC = 1. ! For CASE bounds

These are used to obtain and convert task values

DBGEXTSK_CVT_VALUE_NUM = 1.
DBGEXTSK_CVT_NUM_VALUE = 2.
DBGEXTSK_NEXT_TASK = 3.

These are used to ask ADA to display task information

DBGEXTSK_SHOW_TASK = 4.
DBGEXTSK_SHOW_STATISTICS = 5.
DBGEXTSK_SHOW_DEADLOCK = 6.

These are used to get and set various attributes of one or more tasks

Task state
DBGEXTSK_GET_STATE = 7.
DBGEXTSK_GET_ACTIVE = 8.
DBGEXTSK_SET_ACTIVE = 9.
DBGEXTSK_SET_TERMINATE = 10.
DBGEXTSK_SET_HOLD = 11.

Task priority
DBGEXTSK_GET_PRIORITY = 12.
DBGEXTSK_SET_PRIORITY = 13.
DBGEXTSK_RESTORE_PRIORITY = 14.

Task registers
DBGEXTSK_GET_REGISTERS = 15.
DBGEXTSK_SET_REGISTERS = 16.

These are used to control definable events

DBGEXTSK_ENABLE_EVENT = 17.
DBGEXTSK_DISABLE_EVENT = 18.

DBGEXTSK_MAX_FUNC = 18; ! For CASE bounds

LITERAL

A minimum task code is defined for CASE statement bounds.

DBGEXT\$K_MIN_FUNCT = 1.

CVT_VALUE_NUM takes a task value and converts it to a task number.

INPUT - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

OUTPUT - The task number is returned in the DBGEXT\$L_TASK_NUMBER field.

(If the task does not exist, this function returns
status ST\$K_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%
%(VALUE IS NOT LEGAL OR ACCVIO? -tbs))%

DBGEXT\$K_CVT_VALUE_NUM = 1.

CVT_NUM_VALUE takes a task number and converts it to a task value.

INPUT - The task number is placed in the DBGEXT\$L_TASK_NUMBER field.

OUTPUT - The task value is returned in the DBGEXT\$L_TASK_VALUE field.

(If the task does not exist, this function returns
status ST\$K_SEVERE).%((TASK DOES NOT EXIST CODE? -tbs))%

DBGEXT\$K_CVT_NUM_VALUE = 2.

NEXT_TASK gives a task value and asks ADA to specify the "next" task.
The ordering of tasks is up to the ADA run-time system. The only
requirement on order is that if we start with any task, and repeatedly
ask for the "next" without giving the user program control in between,
then we will cycle through all the tasks and return to the task we
started with. If selection criteria are imposed, then we will cycle
through all tasks which match that criteria.

INPUTS - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

If the TASK_VALUE field is zero (implying the NULL task) the
next task will be the main task of the program.

The ALL flag is ignored, ADA will consider it on by default.

The set of tasks to cycle through can be restricted by
imposing a selection criteria. The PRIORITY, and/or STATE,
and/or HOLD fields can contain values which a task must match
to be part of the set (e.g. SHOW TASK/PRI=3/HOLD/STATE=READY).
When such a restriction is desired, the DBGEXT\$V_xxx SPECIFIED
bits must be set accordingly. If no restriction is desired,
the SPECIFIED bits must be zero. A task must match all the
criteria which are specified to be part of the set.

!%((Multiple PRI and STATE can be given as these are bit fields -tbs))%

```

0494 |
0495 | OUTPUT - The "next" task value is returned in DBGEXT$TASK_VALUE.
0496 |
0497 | DBGEXT$K_NEXT_TASK = 3.
0498 |
0499 |
0500 | SHOW_TASK is used to request that ADA display information about a
0501 | specified task.
0502 |
0503 | INPUTS - The task value is placed in the DBGEXT$TASK_VALUE field.
0504 |
0505 | The address of a print routine that ADA is to call, to display
0506 | the information, is placed in the field DBGEXT$PRINT_ROUTINE
0507 | (see DBG$PRINT_ROUTINE below).
0508 |
0509 | If the DBGEXT$V_FULL bit is set, more detailed information
0510 | is displayed.
0511 |
0512 | OUTPUT - none.
0513 |
0514 | DBGEXT$K_SHOW_TASK = 4.
0515 |
0516 |
0517 | SHOW_STATISTICS requests that the ADA run-time system display
0518 | statistics about the overall state of the multitasking system.
0519 |
0520 | INPUTS - The address of a print routine is given in the field
0521 | DBGEXT$PRINT_ROUTINE.
0522 |
0523 | If the DBGEXT$V_FULL bit is set, more detailed information
0524 | is displayed.
0525 |
0526 | OUTPUT - none.
0527 |
0528 | DBGEXT$K_SHOW_STAT = 5.
0529 |
0530 |
0531 | SHOW_DEADLOCK requests that the ADA run-time system display information
0532 | about deadlocks within the multitasking system.
0533 |
0534 | INPUTS - The address of a print routine is given in the field
0535 | DBGEXT$PRINT_ROUTINE.
0536 |
0537 | If the DBGEXT$V_FULL bit is set, more detailed information
0538 | is displayed.
0539 |
0540 | OUTPUT - none.
0541 |
0542 | DBGEXT$K_SHOW_DEADLOCK = 6.
0543 |
0544 |
0545 | GET_STATE inquires about the "state" and HOLD condition of a task. The
0546 | "state" can be one of RUNNING, READY, SUSPENDED, TERMINATED.
0547 | The state codes are defined below.
0548 |
0549 | INPUT - The task value is placed in the DBGEXT$TASK_VALUE field.
0550 |

```

0551 | OUTPUTS - A code representing the state is returned in the `%((V_STATE - tbs))%`
 0552 | `DBGEXT$W_STATE` field.
 0553 |

0554 | The `DBGEXT$V_HOLD` field is also set if the task is on HOLD.
 0555 |

0556 | `DBGEXT$K_GET_STATE` = 7,
 0557 |

0558 | `GET_ACTIVE` obtains the task value of the active task.
 0559 | (The active task is that task in whose context (stack and register set)
 0560 | `DEBUG` is executing. This is contrasted with the "visible task" --
 0561 | the task whose register set is temporarily in use by `DEBUG`
 0562 | as a default for the purposes of `SHOW CALLS`, `EXAMINE`, etc.).
 0563 |

0564 | INPUTS - none
 0565 |

0566 | OUTPUT - The task value of the active task is returned
 0567 | in `DBGEXT$L_TASK_VALUE`.
 0568 |

0569 | `%((Can the active task be the null task? -tbs))%`
 0570 |

0571 | `DBGEXT$K_GET_ACTIVE` = 8,
 0572 |

0573 | `SET_ACTIVE` requests the run-time system to switch the active
 0574 | task to that given in `DBGEXT$L_TASK_VALUE`. The "long form" `DBG`
 0575 | control block is used. The registers provided by `DEBUG` in the control
 0576 | block are those of the (currently) active task. The run-time
 0577 | system uses these to save the registers of the active task. It
 0578 | may also modify this register set, (currently only the PC and PSL).
 0579 | When this call returns, `DEBUG` should use the possibly-modified
 0580 | register values as the active register set. If the `PSEUDO_GO` bit
 0581 | is set, `DEBUG` should then perform the actions of a normal `GO`,
 0582 | except that ASTs are left disabled. This "pseudo-`GO`"
 0583 | will enter special run-time code that will switch-out the
 0584 | currently active task, switch-in the requested active task, and
 0585 | reinvoked `DEBUG` in that task. (A special event code is assigned
 0586 | to this "reinvoke `DEBUG` event". The reinvocation event signifies
 0587 | to `DEBUG` that certain components of its state are to be
 0588 | gotten from values saved from `DEBUG`'s prior incarnation, not those
 0589 | at the reinvocation event. One such saved state component is
 0590 | the "AST enablement" status - whether ASTs were enabled when
 0591 | `DEBUG` was invoked.)
 0592 | Despite these gyrations, to the user typing
 0593 | `DBG> SET TASK/ACTIVE T1`, it appears he has entered a simple command
 0594 | immediately followed by a `DBG>` prompt.
 0595 |

0596 | INPUTS - The task value of the to-become-active task is set
 0597 | in `DBGEXT$L_TASK_VALUE`.
 0598 |

0599 | The registers of the (currently) active task are stored in
 0600 | fields `DBGEXT$L_R0` through `DBGEXT$L_PSL`.
 0601 |

0602 | OUTPUTS - The register set of the new active task, as
 0603 | modified by the run-time system, in `DBGEXT$L_R0`
 0604 | through `DBGEXT$L_PSL`.
 0605 |
 0606 |
 0607 |

The DBGEXT\$V_PSEUDO_GO flag may be set, in which case,
DEBUG should perform a "pseudo go" operation.

DBGEXT\$K_SET_ACTIVE = 9.

SET_TERMINATE is used to cause ADA to terminate a task. It is used
to implement the command SET TASK/TERMINATE.

INPUTS - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

If the TASK_VALUE field is zero and the ALL flag
is set, then the function is done for all tasks.

OUTPUT - none

DBGEXT\$K_SET_TERMINATE = 10.

SET_HOLD is used to put a task on hold or to release a task that was
previously put on hold. It is used to implement the command
SET TASK/HOLD which leaves the state of a task as-is, except that each
task is marked HOLD.

INPUTS - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

If the TASK_VALUE field is zero and the ALL flag
is set, then the function is done for all tasks.

%((Will the /ALL selection criteria be used for the SET_xxx codes? -tbs))%

The desired status of HOLD is placed into the
DBGEXT\$V_HOLD field. (1 => HOLD, 0 => RELEASE)

%((Is the request 1=>1 or 0=>0 legal? -tbs))%

OUTPUT - none

DBGEXT\$K_SET_HOLD = 11.

GET_PRIORITY inquires about the priority of a specified task.

INPUT - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

OUTPUT - The priority is returned in the DBGEXT\$W_PRIORITY field.

DBGEXT\$K_GET_PRIORITY = 12.

SET_PRIORITY is used to set the priority of a specified task.

INPUTS - The task value is placed in the DBGEXT\$L_TASK_VALUE field.

If the TASK_VALUE field is zero and the ALL flag
is set, then the function is done for all tasks.

0665 0
 0666 0
 0667 0
 0668 0
 0669 0
 0670 0
 0671 0
 0672 0
 0673 0
 0674 0
 0675 0
 0676 0
 0677 0
 0678 0
 0679 0
 0680 0
 0681 0
 0682 0
 0683 0
 0684 0
 0685 0
 0686 0
 0687 0
 0688 0
 0689 0
 0690 0
 0691 0
 0692 0
 0693 0
 0694 0
 0695 0
 0696 0
 0697 0
 0698 0
 0699 0
 0700 0
 0701 0
 0702 0
 0703 0
 0704 0
 0705 0
 0706 0
 0707 0
 0708 0
 0709 0
 0710 0
 0711 0
 0712 0
 0713 0
 0714 0
 0715 0
 0716 0
 0717 0
 0718 0
 0719 0
 0720 0
 0721 0

The desired priority is placed in the DBGEXT\$W_PRIORITY field.

OUTPUT - none.

DBGEXT\$K_SET_PRIORITY = 13,

RESTORE_PRIORITY is used to restore the priority of a task back to its normal value (as it would be without DEBUG intervention).

INPUTS - The task value is placed in the DBGEXT\$T_TASK_VALUE field.

If the TASK_VALUE field is zero and the ALL flag is set, then the function is done for all tasks.

OUTPUT - none.

DBGEXT\$K_RESTORE_PRIORITY = 14,

GET_REGISTERS is used to obtain the register set of a task.

INPUT - The task value is placed in the DBGEXT\$T_TASK_VALUE field.

OUTPUTS - The register values are returned in the DBGEXT\$T_R0 through DBGEXT\$T_PSL fields.

NOTE: Only DEBUG knows the register set of the active task hence, this call is invalid for the active task. A return status of ST\$K_SEVERE is returned.

DBGEXT\$K_GET_REGISTERS = 15,

SET_REGISTERS is used to change the register values of a task. This may be needed, for example, in SET TASK T;DEPOSIT R5 = 0;60

INPUTS - The task value is placed in the DBGEXT\$T_TASK_VALUE field.

The register values are placed in the DBGEXT\$T_R0 through DBGEXT\$T_PSL fields.

OUTPUT - none.

NOTE: Only DEBUG knows the register set of the active task hence, this call is invalid for the active task. A return status of ST\$K_SEVERE is returned.

DBGEXT\$K_SET_REGISTERS = 16,

ENABLE_EVENT is used during processing of a "SET BREAK/EVENT=" or "SET TRACE/EVENT=" command to enable reporting of a given kind of event.

INPUTS - The DBGEXT\$T_EVENT_ID field contains a code identifying the event being enabled. The possible values of this code are defined below.

J 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1

Page 18
(10)

The DBGEXT\$TASK_VALUE field contains a task value further qualifying the event being enabled. This may be zero if the "ALL" flag is lit.

For example,
if we are enabling "task termination" and we supply a task value, then we only want to break on termination of that task. If we enable "task termination" events and set the ALL flag, we want to be notified of any task termination.

OUTPUT - none

DBGEXT\$K_ENABLE_EVENT = 17,

DISABLE_EVENT is used during processing of a "CANCEL BREAK/EVENT=" or "CANCEL TRACE/EVENT=" command to disable reporting of a given kind of event.

INPUTS - The DBGEXT\$EVENT_ID field contains a code identifying the event being disabled. The possible values of this code are defined below.

The DBGEXT\$TASK_VALUE field contains a task value further qualifying the event being disabled. This may be zero if the "ALL" flag is lit.

OUTPUT - none

DBGEXT\$K_DISABLE_EVENT = 18,

A maximum task code is defined for CASE statement bounds.

DBGEXT\$K_MAX_FUNCT = 18;

COMPLETION STATUS

The run time system has two means of providing a completion status -- the return value of the function and the contents of DBGEXT\$STATUS.

Function Return Value --

The run time system should, as its first action, attempt to read and verify the field DBGEXT\$V.FACILITY_ID in DBGEXT\$CONTROL_BLOCK. Optionally, it may also PROBE the control block for read/writability. If the FACILITY_ID is correct, the run time system should eventually return:

ST\$K_SUCCESS - service successfully completed

Otherwise, the run time system should immediately return:

ST\$K_SEVERE - service failed

This helps to insure that an incorrect External Control Block will be detected before it is written to.

Contents of DBGEXT\$STATUS --

All other status and error conditions will be placed in the STATUS field of the control block. The possible values of the STATUS field are a composite of severity level and message number. Only two severity values are used. They are given by ST\$V.SEVERITY:

ST\$K_SUCCESS - service successfully completed

In this case the message number (ST\$V.MSG_NO) is zero.

ST\$K_ERROR - service failed

In this case the message number (ST\$V.MSG_NO) is one of the following:

LITERAL

DBGEXT\$K_FUNCTION_NOT_IMP = 0,

The function requested is not implemented by the facility.

DBGEXT\$K_TASK_NOT_EXIST = 1,

Task number cannot be translated to a task value because the task does not exist. Or task value does not point to a currently existing task (this cannot always be detected).

DBGEXT\$K_TASK_IS_ACTIVE = 2,

Returned on a SET_REGISTER or GET_REGISTER function for the active task. The run time system cannot access the registers of the active task.

DBGEXT\$K_TASK_IS_NULL = 3;

L 16
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255SDUA28:[DEBUG.SRC]DBGEXT.REQ;1 Page 20
(11)

: 0817 0
: 0818 0

! Returned on a SET_ACTIVE function for the null task.

PRINT ROUTINE INTERFACE

The following defines how to use the DEBUG print routine whose address is given in the DBGEXT\$SL_PRINT_ROUTINE field.

BIND
DBG\$PRINT_ROUTINE = .control_block [DBG\$SL_PRINT_ROUTINE];

DBG\$PRINT_ROUTINE (NEW_LINE,
STRING_TO_PRINT,
FAO_ARG_1,
FAO_ARG_2,...
FAO_ARG_n) : NOVALUE

NEW_LINE - this can have one of two values:

- 0 - Place the given string in the output buffer.
- 1 - If the given string is non-zero, first place it in the buffer. In all cases, output the buffer to the screen.

STRING_TO_PRINT

- this is a pointer to a counted ascii string
E.g., UPLIT (%ASCIC 'Output this text')
This may be zero if the ACTION_CODE is 'NEWLINE'.

There may be FAO arguments following the string.
The string thus may contain embedded FAO commands
such as '!AC', '!SL', and so on.

%((FIXUP - THIS EXTENSION IS NOT GOOD!! %((%))%)%

In addition, there will be a DEBUG-specific extension
to FAO which can be used for symbolizing addresses.
There will be a new command '!SA' for "symbolize address".
This indicates that the corresponding FAO argument
is an address. It's symbolization is to be embedded into
the string.

FAO_ARG1 through FAO_ARGn - optional parameters for FAO arguments.

Example: suppose FOO\L is located at address 200. Then:

DBG\$PRINT_ROUTINE (DBGEXT\$K_NEWLINE,
UPLIT (%ASCIC 'Task switch at location !SA'),
200);

This would output:

"Task switch at location FOO\L"

EVENT ID

The following define the possible values of the DBGEXT\$EVENT_ID field.
These are the predefined events that we can break or trace on.

LITERAL

DBGEXT\$K_MIN_EVENT_CODE = 0.

DBGEXT\$K_INVOKE_DEBUG = 0. ! Unconditional DEBUG invokation

DBGEXT\$K_TASK_ACTIVATION = 1. ! First transition of a task to RUNNING

DBGEXT\$K_TASK_SUSPENSION = 2. ! Transition from RUNNING to SUSPENDED

DBGEXT\$K_TASK_SWITCH_FROM = 3. ! Transition from RUNNING to some state

DBGEXT\$K_TASK_SWITCH_TO = 4. ! Transition from some state to RUNNING

DBGEXT\$K_TASK_TERMINATION = 5. ! Any kind of termination

! Ada specific tasking codes:

DBGEXT\$K_TASK_ABORT_TERM = 6. ! Termination by abort

DBGEXT\$K_TASK_EXCEP_TERM = 7. ! Termination by unhandled exception

DBGEXT\$K_TASK_EXCEP_REND = 8. ! Exception propagating out of rendezvous

DBGEXT\$K_TASK_ENTRY_CALL = 9. ! Executing an entry call

DBGEXT\$K_TASK_ACCEPT = 10. ! Executing an accept

DBGEXT\$K_TASK_SELECT = 11. ! Executing a select

DBGEXT\$K_MAX_EVENT_CODE = 11;

EVENT CONTROL BLOCK

The Event Control Block is the data structure that the ADA (or other) facility passes to DEBUG when it signals that a given event has occurred.

For example, if you do a SET BREAK/ADAEVENT=TASK_SWITCH_TO, then when a task switch occurs, the ADA run-time system will signal the special signal DBGS_EVENT. A pointer to an "Event Control Block" is passed as the "FAO argument" of DBGS_EVENT. (E.g., LIBSSIGNAL (DBGS_EVENT, 1, .EVENT_CONTROL_BLOCK). (Note that this condition cannot properly be an SSS condition because they are not allowed to have FAO arguments other than PC and PSL (except for the hardware conditions). Hence, the facility DBG was chosen. This condition is a DEBUG-defined condition that anyone can signal. The FAO count of 1 is required so that the message conforms to a legal format for a message vector.) Through proper use of the SEVERITY field and the NOMESSAGE bit in the condition, the signaller can be assured that events will be "reflected" by Traceback should DEBUG not be mapped into the image (for some reason). So there really are no restrictions on when this condition can be signalled.

The control block contains a code indicating the facility that has originated the event and another code to indicate what event has occurred. It also contains message text to be output announcing the event.

The following illustrates the Event Control Block:

0	DBGEXT\$L_EVNT_FACILITY_ID
1	DBGEXT\$L_EVNT_EVENT_ID
2	DBGEXT\$L_EVNT_FLAGS
3	DBGEXT\$L_EVNT_MESSAGE_TXT
4	DBGEXT\$L_EVNT_ARG_COUNT
5	DBGEXT\$A_EVNT_ARG_VECT
	...

FIELD DBGEXT\$EVNT_FIELDS =

```

SET
DBGEXT$L_EVNT_FACILITY_ID = [0, 0, 32, 0],
DBGEXT$L_EVNT_EVENT_ID   = [1, 0, 32, 0],
DBGEXT$L_EVNT_FLAGS      = [2, 0, 32, 0],
DBGEXT$V_EVNT_MORE_TEXT  = [2, 0, 1, 0],      ! Flag bit 0
DBGEXT$V_EVNT_REENTRY    = [2, 1, 1, 0],      ! Flag bit 1

DBGEXT$L_EVNT_MESSAGE_TXT = [3, 0, 32, 0],
DBGEXT$L_EVNT_ARG_COUNT   = [4, 0, 32, 0],
DBGEXT$A_EVNT_ARG_VECT    = [5, 0, 0, 0]
TES:

```

LITERAL
DBGEXT\$K_EVNT_BASE_SIZE = 5;

MACRO
DBGEXTSEVENT CONTROL BLOCK(NUM ARGS) =
BLOCK [DBGEXT\$K_BASE_SIZE * NUM_ARGS ,LONG]
FIELD (DBGEXTSEVENT_FIELDS)%;

Explanation of fields:

FACILITY_ID field: The code for the facility signaling the event. If the CUST_DEF bit is set the event is a "user event". Otherwise, the only supported codes are ADA, PPA, and scan.

EVENT_ID field: This field contains the event code. Event codes are numbered from 1 within each facility. Event code 0 is reserved in all facilities. It represents the unconditional event, that is, unconditional DEBUG entry. If the EVENT_ID field is zero, the REENTRY bit is checked.

MESSAGE_TXT field: This is a pointer to a counted ascii string. The string represents a message to be printed when the event occurs and is formatted as an "fao control string". The string may take FAO arguments. The string may also contain the DEBUG extension to FAO, '!SA', in order to symbolize an address. This extension is described above. NOTE: if this field is 0, it indicates that there is no message.

ARG_COUNT field: Count of the number of FAO arguments that go with the text.

ARG_VECT field: A vector of FAO arguments.

MORE_TEXT flag: If this flag is TRUE, it indicates that DEBUG is to return control at the point of the signal after displaying the message. This is to be used for output of multi-line messages. (I.e., the run-time system should then resignal the event with the next line of message text in the MESSAGE_TXT field).

REENTRY flag: If this flag is TRUE, then this event is a DEBUG-reentry event that has occurred after a PSEUDO GO. DEBUG is thereby instructed to restore certain components of its state from the values they had at DEBUG's last incarnation (e.g. AST enablement). For this flag to be checked by DEBUG, the EVENT_ID field MUST BE ZERO, thus indicating

E 1
15-Sep-1984 23:02:11
15-Sep-1984 22:42:35

VAX-11 Bliss-32 V4.0-742
_S255\$DUA28:[DEBUG.SRC]DBGEXT.REQ;1

: 1014 0
: 1015 0

unconditional entry to DEBUG.

REGISTERING EVENTS WITH DEBUG

DEBUG's event handling feature is available to user programs as well as Digital software. DEBUG maintains an event table for each facility that chooses to register its events with DEBUG.

Registering an event with DEBUG is very simple. The facility need only signal the following signal after DEBUG has been invoked in an image:

```
LIB$SIGNAL(DBG$ REGISTER_EVENTS,
            first_event_condition,
            second_event_condition,
            etc.
```

A list of event conditions is chained below a master condition of DBG\$ REGISTER_EVENTS. This signal may be raised as many times as desired to add more events to DEBUG's event table. Since DEBUG derives the facility number from the event condition, events for different facilities may be registered with the same signal.

The event conditions appearing in the message vector must be defined in the facilities message file. The string defined in the message file is the string that DEBUG will use to name the event.

For example, suppose we wish to add an event of PLI\$ TASK_SWITCH. The following would do it:

1. Add to PLI's message file:
PLI\$ FACILITY = xxx
TASK_SWITCH "TASK_SWITCH"
2. Register the event with DEBUG
LIB\$SIGNAL(DBG\$ REGISTER_EVENTS, PLI\$ TASK_SWITCH)

After the registration, any user can then type
SET BREAK/EVENT=PLI\$ TASK_SWITCH

A command SET EVENT/FACILITY='PLI\$ ' can be used so the facility prefix can be omitted, e.g. SET BREAK/EVENT=TASK_SWITCH. This will then not be confused with an Ada task switch. SET EVENT/NOFACILITY will eliminate the automatic prefixing of event names.

To simplify the registration of events by facilities, any facility should provide an entry point that users can call from the DEBUGGER to load the events of that facility. To load PLI's events, then, a user would merely type

```
DBG> CALL PLI$ LOAD_EVENTS
```

** Obviously, Ada's events should be registered with this same general mechanism

COMMAND QUALIFIERS

:
: BLISS/LIBRARY=LIB\$:DBGEXT.L32/LIST=LISS:DBGEXT.LIS SRC\$:DBGEXT.REQ
: Run Time: 00:06.5
: Elapsed Time: 00:09.6
: Lines/CPU Min: 9962
: Lexemes/CPU-Min: 15106
: Memory Used: 38 pages
: Library Precompilation Complete

0083 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0084 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

